# 基于 FIUT 的并行频繁项集增量更新算法 *

张　航，张　欣†，张平康，李　琪

(贵州大学 大数据与信息工程学院，贵阳 550025)

**摘　要：** 针对目前大数据快速增加的环境下，海量数据的频繁项集挖掘在实际中所面临的增量更新问题，在频繁项超度量树算法（frequent items ultrametric trees，FIUT）的基础上，引入 MapReduce 并行编程模型，提出了一种针对频繁项集增量更新的面向大数据的并行算法。该算法通过检查频繁超度量树叶子节点的支持度来确定频繁项集，同时采用准频繁项集的策略来优化并行计算过程，从而提高数据挖掘效率。实验结果显示，所提出的算法能快速完成扫描和更新数据，具有较好的可扩展性，适合于在动态增长的大数据环境中进行关联规则相关数据挖掘。

**关键词：** 大数据；频繁项集；MapReduce；增量更新；频繁项超度量树

**中图分类号：** TP301.6　　doi: 10.3969/j.issn.1001-3695.2017.12.0854

## Incremental updating algorithm of parallel frequent itemsets based on FIUT

Zhang Hang, Zhang Xin†, Zhang Pingkang, Li Qi

(*College of Big Data & Information Engineering Guizhou University*, *Guiyang* 550025, *China*)

**Abstract:** With the rapid increase in the big data environment, frequent itemsets data mining faces in the actual incremental update problem. This paper proposes a parallel incremental updating algorithm based on MapReduce for frequent itemsets in frequent items ultrametric trees. The algorithm utilizes the support of frequent check ultrametric tree leaf node to determine the frequent itemsets and frequent itemsets using quasi strategies to optimize the parallel computing process, so as to improve the efficiency of data mining. According to the compared experiment results, it shows that the proposed algorithm is able to scan and update data efficiently, and has good scalability. It can be used for mining association rules in the incremental big data environment.

**Key words:** big data; frequent itemsets; Mapreduce; incremental updating; frequent items ultrametric trees



## 0　引言

关联规则的数据挖掘作为数据挖掘的重要方面之一，主要用于对大量数据中项集进行相关分析，是数据挖掘领域重要的研究方向之一[1]。在目前大数据的实际应用中，单机平台面临海量数据在存储能力、处理能力和挖掘上都面临显而易见的瓶颈。而通过并行化的方式，不仅能有效解决内存不足的问题，还能极大地提升挖掘效率。另一方面，当新数据增加后，或会导致原始数据处理结果不再准确，而在新的数据全集上重新运行会耗费大量的计算资源。因此，将新增数据集和已知处理结果完成增量更新式的数据处理，可以极大地提升数据处理效率。综上，将并行处理的思想与增量更新的方法进行结合利用具有重大的现实意义。

随着现在数据量爆炸式的增长，并行、分布式的关联规则挖掘算法越来越成为学界、实务界的关注热点。结合 MapReduce 编程模型，研究者们从减少 MapReduce 任务数、候选项集数量、并行节点之间的通信量等多个角度对并行化算法效率进行了提升[2]。Li 等人提出了并行 FP-Growth 算法，实现了 FP-Growth 算法在 MapReduce 计算框架上的移植[3]。杨勇等人提出基于 MapReduce 的并行增量更新算法，通过引入分块索引提高挖掘效率，同时采用负载均衡策略保证集群节点之间的负载均衡[4]。

针对海量数据的关联规则挖掘过程中存在的增量更新问题，本文基于 MapReduce 计算框架设计并实现了一种 FIUT 关联规则增量更新算法。该算法通过对传统 FIUT 算法进行并行化改进，将其移植到 MapReduce 计算框架上，使其在分布式环境下完成对 FIU-tree 的增量更新，同时结合准频繁项集的思想减少扫描原始事务数据库的需要，提高了并行计算效率。文章最后，通过实验证明了该算法的有效性和可扩展性。

**收稿日期：** 2017-12-18；**修回日期：** 2018-01-31　　**基金项目：** 国家国际科技合作专项项目（2014DFA00670）；贵州省研究生教育教学改革重点课题（黔教研合 JG 字［2016］15）；贵州省科技厅工业攻关项目（黔科合 GY 字［2010］3056）

**作者简介：** 张航（1991-），男，四川成都人，硕士研究生，主要研究方向为云计算与大数据等；张欣（1976-），男（通信作者），贵州贵阳人，副教授，博士，主要研究方向为下一代无线通信及应用等（2225400115@qq.com）；张平康（1993-），男，安徽马鞍山人，硕士研究生，主要研究方向为图像处理等；李琪（1990-），男，贵州贵阳人，硕士研究生，主要研究方向为并行化计算等.

## 1 相关概念

### 1.1 FIUT 算法

FIUT 算法是由 Tsay 等人 2009 年提出的一种用于在数据库中的频繁项集挖掘的算法[5]。其算法基本类似于 FP-Growth 算法，都是通过两次扫描数据库以得到频繁项集，使用事务项聚类来减小搜索空间，只有事务项中的频繁项集才能插入到 FIU-tree 上以压缩存储，但所有的频繁项的挖掘与 FP-Growth 算法通过递归挖掘不同，其都是通过检查 FIU-tree 上的叶子节点的计数得到。FIUT 算法的主要思想及步骤可分为三个步骤，分别是生成频繁 1-项集和 k-itemsets;建立 k-FIU-tree;挖掘 k-FIU-tree。

### 1.2 准频繁项集

准频繁项集(pre-large itemsets)是指未达到频繁项集要求，但在将来有可能会成为频繁项集的项集[6]。它引入了两个支持度阈值：支持度阈值下限 $S_L$ 和支持度阈值上限 $S_U$，通过这两个阈值将原始事务数据库增量更新后的所有项集分为九种类别：$C_1,C_2,C_3,C_4,C_5,C_6,C_7,C_8,C_9$，如图 1 所示。

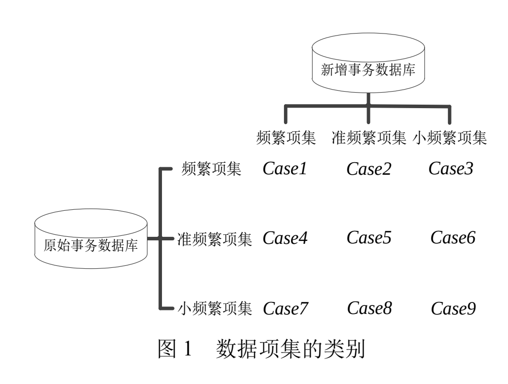


图 1 数据项集的类别

其中，类别 $C_1,C_5,C_6,C_8,C_9$ 都不会造成事务数据库频繁模式的变化；类别 $C_2$ 和 $C_3$ 有可能会去除已存在的关联规则；类别 $C_4$ 和 $C_7$ 有可能会增加新的关联规则。

### 1.3 MapReduce 编程模型

MapReduce 是 Apache 旗下的用于处理和产生大数据集的并行计算编程模型。其主要原理是利用一个输入的键值对集合，通过一系列处理来产生一个输出的键值对集合。它将对数据集的操作分解为 Map 任务和 Reduce 任务，用户可以通过重写 mapper 和 reducer 这两个函数分发给集群上的各个节点来对数据进行处理。每个 Map 任务将输入的数据分片分成若干个键值对<key,value>的形式，mapper 函数每次处理一个键值对，处理后就会输出一个同样形式的<key,value>数据，之后 MapReduce 会将属于同一个键的值分发到一起生成中间结果作为 Reduce 的输入数据；而 Reduce 任务则是将所得到的具有相同 key 值的 value 集合交给 reducer 函数进行处理并输出最终的计算结果[7]。

## 2 FIUT 增量更新算法

本文主要针对关联规则的海量数据挖掘和增量更新问题，提出了基于 MapReduce 的 FIUT 关联规则增量更新算法。该算法将求取频繁项集的操作主要分为两个步骤：a)针对原事务数据库 DB 进行数据分组，构建各映射数据库，利用 MapReduce 并行挖掘产生局部的频繁项集和准频繁项集并进行保存，合并构建全局候选项集；b)针对新增事务数据库进行相应的增量更新，利用之前的挖掘结果合并新增事务数据库进行挖掘，完成频繁项集的增量更新。

### 2.1 原始数据库挖掘

设 $I=\{I_1,I_2,\ldots,I_m\}$代表事务项数据库中所有项的集合。原始事务数据库设为 $DB=\{P_1,P_2,\ldots P_n\}$，其中 Pi 代表其中的第 i 条事务记录，它作为事务项集合 I 的一个项目子集，|DB|代表原始事务数据库中事务记录的数目。新增事务数据库设为 db，|db|代表新增事务数据库中事务记录的数目。U 为增量更新后的整个事务数据库(即 DB *U* db)。$S_L$ 为准频繁项目的支持度阈值下限，$S_U$ 为准频繁项目的支持度阈值上限，其中 $S_U>S_L$。

**输入：** 数据库 DB，支持度阈值下限 $S_L$，支持度阈值上限 $S_U$。

**输出：** 数据库 DB 的频繁项集和准频繁项集，数据库 DB 的 1-项集，数据库 DB 的头表分组。

**a**）原始事务数据库项目支持度计数。在 Map 阶段，每个 Map 任务读取先事务数据库的数据，之后以 HDFS 文件分片为单位对事务数据进行处理，mapper 函数输入为< key=offset，value=$P_i$ >，其中，offset 是一条事务 $P_i$ 在该分片的偏移量。Mapper 函数的输出为<key=$a_{ij}$ ,value=1>，其中 $a_{ij}$ 是 Pi 中的一项。经过 Shuffle 阶段完成相同项的聚集和分发，在 Map 任务进行完数据处理后，所有的 key=$a_{ij}$ 的键值对将分配到相对应的 Reduce 任务上进行下一步。在 Reduce 阶段，reducer 函数将输入到其中的<key=$a_{ij}$ ,value = {1,1,1,…,1} >的形式进行求和，统计输出相对应的频次，所有结果合并后得到数据库 DB 的 1-项集 $C_1$。

**b)**分组生成。读取步骤 a）产生的 1-项集 $C_1$，将每一项同支持度阈值下限 $S_l$ 和支持度阈值上限 $S_u$ 比较，得到 1-频繁项集 $F_1$ 和 1-准频繁项集 $PF_1$，修剪不满足阈值要求的项，然后将 $F_1$ 和 $PF_1$ 中的各项分为 n 个组别，每组别都对应一个唯一的编号 id 用以互相区分，标记所有项所对应的分组号，其分组记为 G-list。

**c)**并行挖掘频繁项集和准频繁项集。这一过程通过第二次 MapReduce 任务完成。每一个 Map 任务先读取步骤 b)所产生分组 G-list，这里的 G-list 表使用 Hash 结构，它的每一个项都映射到其所在的事务数，然后根据 G-list 读取输入的事务数据库数据分片 $DB_i$，每条事务 $P_i$ 都会被分到对应的分组中，这时 map 函数就会输出多个键值对数据<key=id ,value = $P_i$>，键值对中所有属于同一个分组的事务记录将会被分发到同一个 reducer 函数中，其输入为< key=id , value = $\{DB_{r1}, DB_{r2}, DB_{r3}, \ldots, DB_{rk}\}$>。之后每个 Reduce 任务根据其分组的 h-项集和映射数据库$\{P_1,P_2,\ldots P_n\}$将其包含的每一项映射到相应的组在本地构

建条件 k-FIU-tree，进而挖掘出每个分组的频繁项集和准频繁项集，最后将每个分组的挖掘结果进行合并得到全局的频繁项集和准频繁项集。在构建完成每个分组的 k-FIU-tree 之后，将其序列化存储到 HDFS 文件系统中，以备后续的增量更新[8]。

如图 2 所示，通过上述三个步骤，完成对原始事务数据库 DB 的挖掘，计算出当前数据库 DB 中所有的频繁项集和准频繁项集，同时记录并存储了其每个项目的支持度和每个分组的 k-FIU-tree。

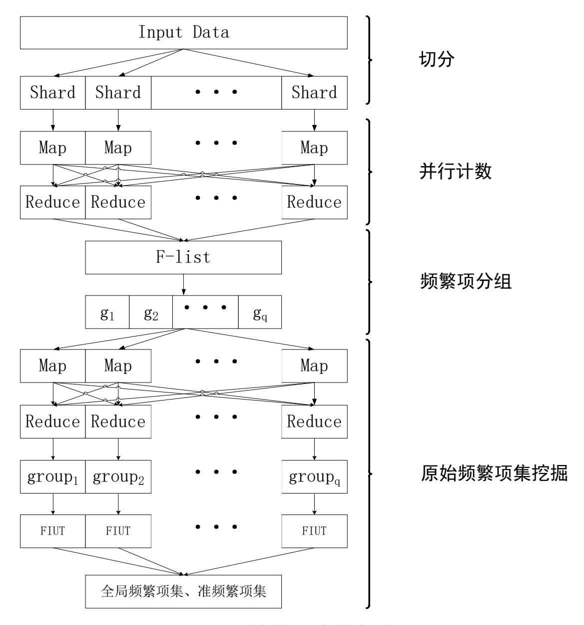


图 2　原始数据库挖掘流程图

### 2.2　新增事务数据库的增量更新

为了方便迭代增量更新及数据计算的统一性，设在增量更新关联规则挖掘的过程中，其新增数据库 db 也分散地存储在集群上的各个节点上。此时其增量更新的主要步骤如下：

**输入：**事务数据库 DB 及其所有频繁项集和准频繁项集，新增事务数据库 db，支持度阈值下限 $S_L$，支持度阈值上限 $S_U$.

**输出：**增量更新后的频繁模式。

**a)** 新增事务数据库项目支持度计数。通过进行一次 MapReduce 任务来读入新增事务数据库 db，统计新增数据库 db 中的支持度计数。其 Map 任务和 Reduce 任务阶段同 2.1 节的步骤 a)类似，首先定位输入的事务记录文件到新增记录行，然后在 mapper 函数中对新增事务数据库进行扫描，之后通过 reducer 函数统计求和得到每个项目在 db 中的支持度计数，也即 db 的 1-项集 $C_1$'。

**b)**分组调整。读取步骤 a)生成的结果 $C_1$'，根据支持度阈值上下限将新增事务集分别划分为三部分，根据 DB 和 db 事务数据库项目的频繁关系对 db 中的项目进行分组。设 Insert_Items，Delete_Items，Rescan_Items 为三个集合，集合中的元素为 db 中项集的子集。其中 Insert_Items 中项集 I 在增量更新后为数据库 U 的频繁项集；Delete_Items 中项集 I 在原始事务数据库 DB 中原为频繁项集，但在新增事务数据库 db 中为准频繁项集或小频繁项集，在增量更新后为数据库 U 的非频繁项集，是需要删除已存在的数据项集；Rescan_Items 可以看做 Insert_Items 的子集，其项集 I 在原始数据库 DB 中为准频繁项集或小频繁项集，在增量更新后数据库 U 中为频繁项集，是需要新增的数据项集。整个步骤主要是对 HDFS 分布式文件系统中原始事务数据库 DB 和新增事务数据库 db 进行扫描，根据他们之间的频繁关系分别放入上述三个集合中。

**c)**增量更新每个分组中的 k-FIU-tree 并挖掘增量后的频繁项集。首先通过一个 mapper 函数对原始事务数据库 DB 和新增事务数据库 db 进行扫描，找到需要增量更新的事务，并分发至特定的分组当中，这样属于同一分组的事务记录就将分发到相同的 reducer 函数中[9]。然后在每个分组的本地更新其分组 k-FIU-tree，并挖掘出每个分组 k-FIU-tree 的新的频繁项集，最后将更新后的分组 k-FIU-tree 合并得到增量更新后的全局结果并序列化存储到 HDFS 文件系统中。

如图 3 所示，通过上述三个步骤，完成在 MapReduce 环境下新增事务数据库的扫描计数，对 k-FIU-tree 进行增量更新，并挖掘出增量后的频繁项集。

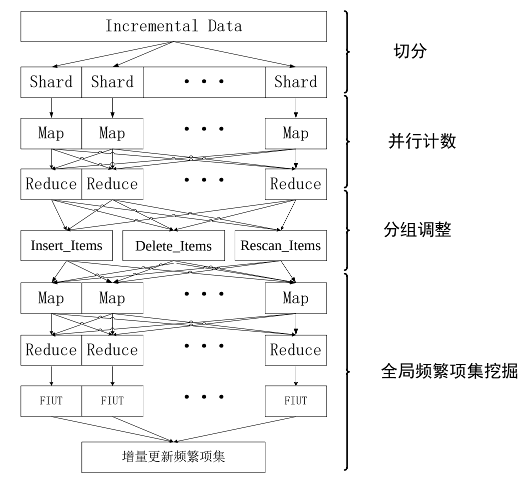


图 3　增量更新数据挖掘流程图

## 3　实验结果分析

### 3.1　实验数据与运行环境

本文实验采用 Frequent Itemset Mining Dataset repository 上所提供的 Webdocs.dat.gz 作为实验数据集[1}{0]，该数据集大小约为 1.48 GB，它包含了 1 692 082 个事务项的共 5 267 656 条记录，最大的事务长度约为 71.472。实验中将数据随机的分为 5 组，每组数据大小约为 300 MByte，每次递增 20%（约 340 000 条事务项）作为其新增事务数据库，支持度阈值下限设置为 3%，支持度阈值上限设置为 5%。

本文实验采用的 MapReduce 并行计算环境是由 5 个节点组成的主从式 Hadoop 集群，其中一个节点作为 NameNode 和 JobTracker，其他节点作为 DataNode 和 TaskTracker。每个节点均保持相同的硬件配置和软件配置，其中 CPU 采用 Intel Core i5 处理器,主频 2.5 GHZ，计算机内存大小为 4 GB，硬盘大小为 160GB，操作系统采用 Centos7 版本，Hadoop 采用 2.6.0 版本，

JDK 采用 jdk-7u79-linux-x64 版本，SSH 采用 OpenSSH 5.8 版本，集群使用默认的配置参数。

### 3.2　算法性能分析

在分布式并行计算环境下，对于增量更新后的数据集的规则挖掘常用的有两种形式，一种是将原始事务数据库同新增事务数据库合并，直接对增量更新后的总数据库进行整体的数据挖掘；另一种则是利用已有的数据库挖掘结果，在其基础之上结合新增数据库进行增量更新。其中采用第一种形式的有 Mahout 开源平台所提供的 PFP-Growth 算法，本文并行化算法采用第二种形式。实验对这两种算法进行了比较分析，结果如图 4 所示

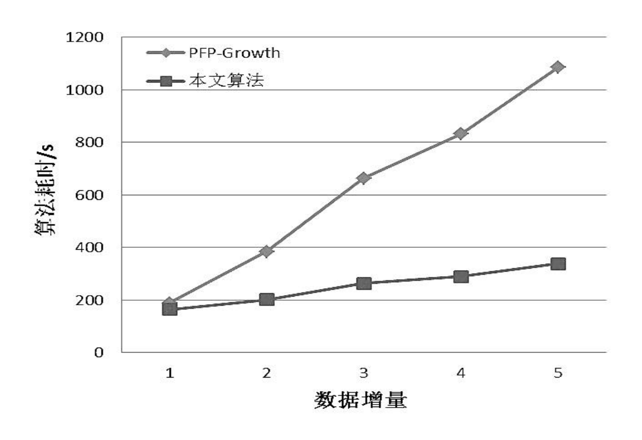



图 4　增量更新性能比较

由图 4 可以看出，在 Hadoop 计算环境下，随着数据增量的不断增加，本文算法所消耗的时间相比于 PFP-Growth 算法更少。由于本文算法在每次增量更新阶段只用对新增事务数据库进行扫描，在很大程度上节省了扫描和统计资源消耗，从而较好地提升了挖掘效率。同时，随着事务数据量的递增，本文算法所消耗的时间开销增幅较为稳定，保持了较好稳定的性能。因此，实验结果证明了本文算法在 Hadoop 环境下能很好地完成数据增量更新任务。

为了验证本文算法的可扩展性，原始事务数据库选为 Webdocs.dat 中 80%的记录（共 4 214 125 条事务项），剩余数据库 20%的记录作为新增事务数据库，支持度阈值下限设置为 3%，支持度阈值上限设置为 5%，基于不同 slave 节点数进行了比较实验，结果如图 5 所示。

由图 5 可以看出，在面对规模较大的数据集，在支持度阈值等条件相同的情况下，数据节点之间交换成本所占比重较低，同时随着 slave 节点数量的增加，数据集分散程度越高，每个节点上处理的子数据集越小，相应每次的迭代时间越短。因此，实验结果证明了本文算法在 Hadoop 环境下具有较好的可扩展性。

## 4　结束语

本文以 FIU-tree 算法为基础，设计并实现了一种基于 MapReduce 编程模型的关联规则并行增量更新算法。本文算法通过查看频繁超度量树叶子节点的支持度确定频繁项集，同时采用准频繁项集的策略来优化并行计算过程，从而提高数据挖掘效率。实验结果表明，本文算法能高效运行于 Hadoop 平台，完成快速扫描和更新数据，具有较好的可扩展性，适合于在动态增长的大数据环境对关联规则进行增量更新。

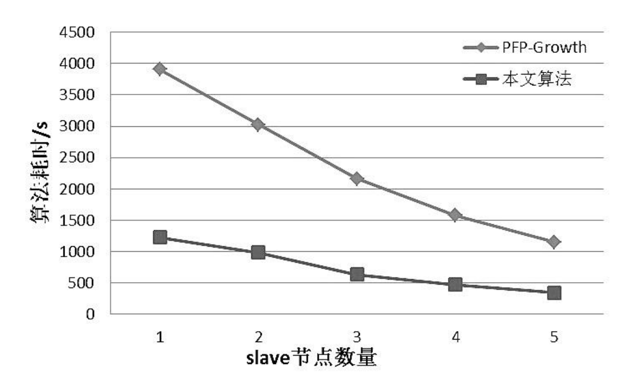



图 5　不同 slave 接点数算法运行时间比较

## 参考文献：

[1] 梁吉业，冯晨娇，宋鹏. 大数据相关分析综述 [J]. 计算机学报, 2016, 39 (1): 1-18.

[2] 肖文，胡娟，周晓峰. 基于 MapReduce 计算模型的并行关联规则挖掘算法研究综述 [J]. 计算机应用研究, 2018, 35 (1): 13-23.

[3] Zhang D, Zhang D, Zhang D, *et al.* Pfp: parallel fp-growth for query recommendation [C]// Proc of ACM Conference on Recommender Systems. New York: ACM Press, 2008: 107-114.

[4] 杨勇，高松松. 基于 MapReduce 的关联规则并行增量更新算法 [J]. 重庆邮电大学学报：自然科学版, 2014, 26 (5): 670-678.

[5] Tsay Y J, Hsu T J, Yu J R. FIUT: A new method for mining frequent itemsets [J]. Information Sciences, 2009, 179 (11): 1724-1737.

[6] Lin C W, Hong T P, Lu W H. Using the structure of prelarge trees to incrementally mine frequent itemsets [J]. New Generation Computing, 2010, 28 (1): 5-20.

[7] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters [C]// Proc of Conference on Symposium on Opearting Systems Design & Implementation. [S. l. ] : USENIX Association, 2004: 10-10.

[8] Mirakhorli M, Clelandhuang J. Detecting, tracing, and monitoring architectural tactics in code [J]. IEEE Trans on Software Engineering, 2016, 42 (3): 205-220.

[9] 施亮，钱雪忠. 基于 MapReduce 的约束频繁项集挖掘算法 [J]. 计算机工程与设计, 2015, 36 (10): 2725-2728.

[10] Frequent Itemset mining dataset repository [DB/OL]. [2012-10-21]. http://fimi. ua. ac. be/data/